

# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-283192

(43)Date of publication of application : 23.10.1998

(51)Int.Cl.

G06F 9/45

(21)Application number : 09-090470

(71)Applicant : HITACHI LTD

(22)Date of filing : 09.04.1997

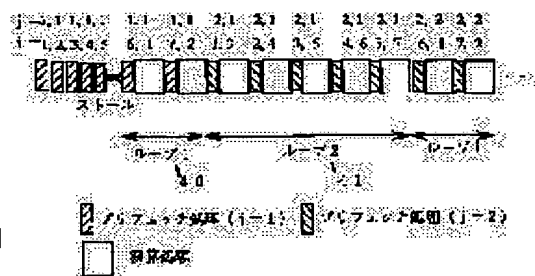
(72)Inventor : TANAKA GIICHI  
ITO SHINICHI  
TSUSHIMA YUJI

## (54) PREFETCHING CODE GENERATION SYSTEM

### (57)Abstract:

**PROBLEM TO BE SOLVED:** To hide a main memory penalty in the case that a data area accessed by a loop exceeds a cache capacity by dividing an inner side loop into two and starting the prefetching of data in a second half loop at the time of the repetition of an outer side loop one before for the data required in a first half loop and in the first half loop for the data required in the second half loop.

**SOLUTION:** The inner side loop is divided into two and a code for incorporating the prefetching processing of the data required at the first part of the repetition processing of the outer side loop one after in the second half loop 41 and incorporating the prefetching of the data required in a second half in the repetition processing of the same outer side loop in the first half loop 40 is generated. For the distribution of the front half loop 40 and the second half loop 41, the processing time of the second half loop 41 is turned to be more than main memory penalty time. In the case that the processing time of the second half loop 41 can not be turned to be more than the main memory penalty, loop development is performed relating to the outer side loop and an arithmetic amount is increased so as to completely hide the main memory penalty.



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

**THIS PAGE BLANK (USPTO)**

[Number of appeal against examiner's decision  
of rejection]

[Date of requesting appeal against examiner's  
decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

**THIS PAGE BLANK (USPTO)**

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-283192

(43) 公開日 平成10年(1998)10月23日

(51) IntCl.<sup>8</sup>

識別記号

F I

G 0 6 F 9/45

G 0 6 F 9/44

3 2 2 G

審査請求 未請求 請求項の数 5 O L (全 10 頁)

(21) 出願番号 特願平9-90470

(22) 出願日 平成9年(1997)4月9日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 田中 義一

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 伊藤 信一

神奈川県横浜市戸塚区戸塚町5030番地 株

式会社日立製作所ソフトウェア開発本部内

(72) 発明者 對馬 雄次

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(74) 代理人 弁理士 小川 勝男

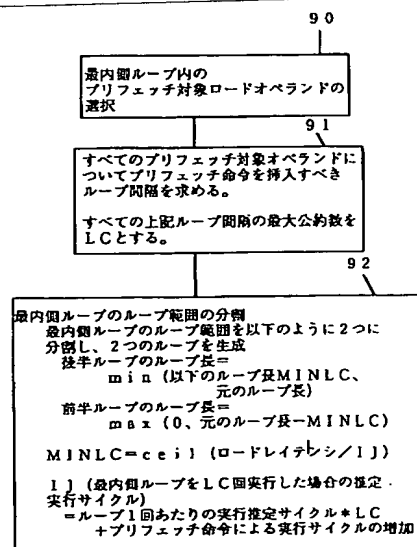
(54) 【発明の名称】 プリフェッチコード生成方式

(57) 【要約】

【課題】 多重ループにおいて、ループでアクセスするデータ領域がキャッシュ容量を超えた場合に、主記憶ペナルティを隠蔽するオブジェクト生成方式を提供する。

【解決手段】 内側ループ処理を2つにわけ、後半ループで、1つ先の外側ループの繰り返しの処理の最初の方で必要となるデータのプリフェッチ処理を組み込み、前半ループで、同一外側ループの繰り返し処理で、後半で必要となるデータのプリフェッチ処理を組み込むコードを生成することで達成される。ここで、ループの分配は後半ループの処理時間が主記憶ペナルティ時間以上となるように行う。

図9



## 【特許請求の範囲】

【請求項1】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの多重ループ部分に対して、上記多重ループのそれぞれがループの実行の直前にはループ実行回数が決まるループの場合、上記多重ループのうち外側ループをループ1、最内側ループをループ2とすると、上記ループ2内にある演算で必要となるデータの一部分を、上記ループ1に関して前のループ繰り返し処理時に、キャッシュに転送するブリフエッチコードを生成することを特徴とするコード生成方式。

【請求項2】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの多重ループ部分に対して、上記多重ループのそれぞれがループの実行の直前にはループ実行回数が決まるループの場合、上記多重ループのうち外側ループをループ1、最内側ループをループ2とすると、上記ループ2を2つのループ繰り返し範囲に分割し、それぞれを前半ループ、後半ループとすると、上記後半ループのコードは、上記ループ1に関して次の繰り返し処理時に上記前半ループの演算の最初に必要となるデータをキャッシュに転送するブリフエッチコードとともとのコードからなり、上記前半ループのコードは、上記前半ループの演算で必要となるデータのうち、上記後半ループでキャッシュに転送されるデータ以外のデータをキャッシュに転送するブリフエッチコードとともとのコードとからなることを特徴とするコード生成方式。

【請求項3】請求項2のコード生成方式において、上記ループ2のループ繰り返し範囲の分割において、上記後半ループの繰り返し数は、上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶アクセスペナルティを超えるように求めた推定ループ繰り返し数か、もともとの上記ループ2のループ繰り返し数のうち小さい繰り返し数を設定することを特徴とするコード生成方式。

【請求項4】請求項3のコード生成方式にて、上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶ペナルティを超える推定ループ繰り返し数が、ある与えられた基準値より、大きい時は、ループ1に関してループ展開ができないか解析を行い、展開可能な場合、展開後の上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶アクセスペナルティを超える推定ループ繰り返し数が、上記基準値より小さくなるまでループ1に関するループ展開を行ったコードを生成することを特徴とするコード生成方式。

【請求項5】請求項2のコード生成方式にて、上記ブリフエッチが必要なループ繰り返し間隔数を求め、上記後半ループのコードは、上記ループ1に関して次の繰り返し処理時に上記前半ループの演算の最初に必要となるデ

ータをキャッシュに転送するブリフエッチコードとともとの処理コードを上記展開数分だけ複製したもののからなり、上記前半ループのコードは、上記前半ループの演算で必要となるデータのうち、上記後半ループでキャッシュに転送されるデータ以外のデータをキャッシュに転送するブリフエッチコードとともとの処理コードを上記展開数分だけ複製したコードからなることを特徴とするコード生成方式。

## 【発明の詳細な説明】

10 【0001】

【発明の属する技術分野】本発明は、命令レベル並列処理を行うプログラムの実行方式に係わり、プログラムがアクセスするデータが大きく、キャッシュに入りきれない場合のループに好適なコード生成方式に関する。

【0002】

【従来の技術】スーパーコンピュータの1つの方向として、スカラプロセッサをノードプロセッサとする並列処理方式が有望視されている。スカラプロセッサを用いた並列スーパーコンピュータが期待されるのは、半導体技術の進歩によるクロック周波数の向上、複数の並列実行可能な演算器を有効に生かすスーバスカラ方式等の命令レベル並列処理の実現により、スカラプロセッサの処理性能が飛躍的に向上しているためである。しかしながら、その高いスカラプロセッサ処理能力は、キャッシュが有効に働くときにのみ達成される。

【0003】スカラプロセッサは、一般的に命令処理装置とキャッシュ及びメモリ装置を有する。スカラプロセッサはメモリにプログラムとデータを格納し、プログラムに記述された命令に従いメモリ中のデータを処理する。キャッシュは命令処理装置からの参照時間の短い比較的小容量の記憶手段であり、プログラム及びデータの一部を一時的に格納する。命令実行にあたり必要なデータはメモリから読み出されるが、同時にデータを含むデータブロックはキャッシュを構成するラインにコピーされ、以後当該ブロック内のデータに対する参照が指定されたときは上記キャッシュのラインからデータを参照する。このメモリからキャッシュラインへのデータブロックの転送をライン転送と呼ぶ。

【0004】命令実行にあたり必要データがキャッシュにない場合、これをキャッシュミスと呼ぶが、キャッシュミスが発生するとライン転送が実行される。従来のスカラプロセッサでは命令実行に伴いこのライン転送が発生すると、その完了まで当該命令の実行が待ち合わされる。従って、キャッシュミスが多発するとライン転送に伴う待ち合わせによりプログラムの実行時間が増大し、スカラプロセッサの処理性能が劣化するという問題があった。特に、大規模科学技術計算では、データ領域が大きく、データの局所性が少ないという性質があるためキャッシュミスによる性能低下は深刻であった。

50 【0005】これに対し、最近では予めプログラムにお

いて、キャッシュミスを引き起こす可能性のある命令に先だってデータの先読みを指示する特殊な命令を実行させることで上記ライン転送に伴う性能劣化を回避する試みがなされている。このデータ先読みをプリフェッチと呼ぶ。例えば、米国IBM社のマイクロプロセッサPowerPCでは指定したオペランドアドレスに位置するデータをキャッシュに読み込むプリフェッチ命令がある。この動作において、キャッシュミスが発生した場合、プリフェッチ命令の完了を待ち合わせることなくライン転送を行う。従って上記データを参照する命令を実行する時には上記データがキャッシュに入っているために上記性能劣化が回避される。

【0006】このようなプリフェッチ命令を利用して、データ転送と演算をオーバラップさせることで、実質的に主記憶レイテンシを隠蔽させるコード生成技術が論文Todd C. Mowry, Monica S. Lam, Anoop Gupta「Design and Evaluation of a Compiler Algorithm for Prefetching」ASPLOS-V, ACM 0-89791-535-6/92/0010/0062 pp.62-73に示されている。

【0007】

【発明が解決しようとする課題】しかしながら、上記の従来のコード生成技術では以下に記すような問題があった。図2に示すようなDO20(20)及びDO10(21)で示される多重ループに対するプリフェッチコードの生成において、内側ループDO10(21)のみしか考慮されていなかった。このことを例を用いて具体的に問題点を示す。

【0008】まず、以下のコード生成で前提としたアーキテクチャの仮定を述べる。対象とするスカラプロセッサはロード/ストア命令が2個、浮動小数点演算が2個、及び整数演算が2個、同時に実行でき、キャッシュラインサイズが32バイト、キャッシュミス時の主記憶ペナルティを75サイクルと仮定する。

【0009】図2のソースプログラムに対して、コンパイラは、まず、ローカリティ解析などによりプリフェッチ対象オペランドを探す。その結果、参照b(i, j)22がプリフェッチ対象オペランドであるとする。オペランドの大きさを8バイトとすると、1回のプリフェッチで、対象とするオペランドを含む連続した32バイトのデータがキャッシュに格納されるため、ループ繰り返しのたびにプリフェッチ命令を実行していたのでは無駄で、この例の場合、4回に1回の割合でプリフェッチ命令を出力すればよい。このため、一般的には、コンパイラのループ構造変換部で、最内側ループ21に関して4倍のループ展開を行う。

【0010】この結果を図3に示す。ここで、ループ21がループ25とループ26に分割されている。ループ26は、ループ展開の余りループである。以後、簡単のため、ループ展開の余りループに関しては省略することにする。そして、通常の最適化、コード生成の後に、ブ

リフェッチコードの生成を行う。

【0011】プリフェッチコードの生成は以下のように行う。まず、ループ25のループ1回あたりの実行サイクル数を推定する。ループ25では、ロード命令が4個、ストア命令が4個、浮動小数点演算が4で、プリフェッチ命令が1であるので、ループ1回あたりの処理サイクル数は5サイクル(=max(ceil((4+4+1)/2), 2/2))と推定できる。ここで、ceil関数は、引数の値を超えない最小の整数を意味する。キャッシュミス時の主記憶ペナルティは75サイクルであるので、ループ処理では、15回(ceil(75/5)=15)後で使用されるデータのプリフェッチを行うコードを生成すると、主記憶ペナルティを隠蔽することができる。

【0012】図4は、図3のコードにプリフェッチ命令を付加したものである。ここでA部30は、ループ繰り返しの最初の方で必要となるデータのプリフェッチを行う部分、B部31は演算処理とループ繰り返しに関して15回後で使用されるデータのプリフェッチ命令を含むループ部分、C部32はループ処理の後半部でプリフェッチ命令を含まないループ部分である。

【0013】図5はその時の実行の様子を模式的に示したものである。横軸が時間、斜線箱がプリフェッチ処理、空箱が演算処理である。ただし、図面の都合上、図4のプログラムとは数値的に一致はしていない。つまり、図4のA部は15回のプリフェッチループであるが、図5では、3回に省略されている。ここで、問題であるのは、A部であり、主記憶アクセスペナルティが顕在化する。これは、特にループ処理回数が小さい場合に問題となる。

【0014】以下では、このコードの性能をみつめる。ループ長がN≡10.0の場合、A部の実行時間は、少なくとも主記憶アクセスペナルティの75サイクル、B部は50サイクル(ループ1回あたり5サイクル×ループ10回実行)、そしてC部は60サイクル(ループ1回あたり4サイクル×ループ15回実行)で計185サイクルで、全体の時間の内40%(75/185×100%)が主記憶待ちのオーバーヘッドとなっている。さらに、ループ長が50と小さい場合には、A部は少なくとも75サイクル、B部はループ長が小さいため実行されず0サイクル、C部は、48サイクル(ループ1回あたり4サイクル×ループ12回実行)で、計133サイクルで、全体の時間の内56%(75/133×100%)が主記憶待ちのオーバーヘッドとなる大きな問題がある。A部で、少なくとも75サイクルと書いたのは、現実の計算機では、同時に処理できるプリフェッチ命令の数には限界があるためである。ここでは、十分大きいと仮定した。

【0015】また、上記で説明したプリフェッチ命令の削減のための内側ループ展開法を用いたコード生成は、以下の問題がある。通常、このようなループの構造変換

はコンパイラの前半部分で行われ、その後、最適化処理が施され、コード生成部でブリフェッチ命令が挿入されると考えられる。こうすると、コンパイラ処理の大半で、ループ展開による大きな中間語を扱わねばならず、長大なコンパイル時間が必要になる。ここでは、ラインサイズ32バイトでの例を示したため、高々4倍の展開であったが、ラインサイズ128バイトの計算機では16倍展開ものループ展開が必要となり、さらに深刻となる。

【0016】本発明の目的は、上記の問題点を解決するブリフェッチ命令を用いたコード生成方式を提供することにある。

【0017】

【課題を解決するための手段】上記の目的は、図6に示すように、内側ループ処理を2つにわけ、後半ループ41で、1つ先の外側ループの繰り返しの処理の最初の方で必要となるデータのブリフェッチ処理を組み込み、前半ループ40で、同一外側ループの繰り返し処理で、後半で必要となるデータのブリフェッチ処理を組み込むコードを生成することで達成される。

【0018】例では、外側ループJ=1の、内側ループI=3, 4, 5, 6, 7の処理の際に、外側ループJ=2の内側ループI=1, 2, 3, 4, 5で必要となるデータをブリフェッチする命令を挿入し、外側ループJ=1の、内側ループI=1, 2の処理の際に、内側ループの後半I=6, 7で必要となるデータをブリフェッチする命令を挿入する。ここで、前半ループ40と、後半ループ41の分配は、後半ループ41の処理時間が主記憶ペナルティ時間以上となるようにループ処理を分割する。

【0019】ここで、内側ループの演算数が小さくループ長が短い場合は、後半ループ41の処理時間を主記憶ペナルティ以上にできない場合がある。この場合、当該ループの処理の時間が、想定した規定値以上のループ長の場合には、完全に主記憶ペナルティを隠蔽できるように、外側ループに関してループ展開を行い、演算量を増加させて、この目的を達成する。

【0020】また、ブリフェッチ命令の削減のための内側ループ展開法を用いたコード生成によるコンパイル時間の増大の問題は、コンパイラ処理の後半のコード生成処理において、コードを複製する方式により解決できる。

【0021】

【発明の実施の形態】以下、本発明のコンパイラにおける実施例を図を参照しつつ説明する。図1にコンパイラ全体の構造を示す。図1のソースプログラム1が、構文解析2によって中間語3に変換される。ループ構造変換部4は、これを入力として、多重ループに関するブリフェッチコードの主記憶レイテンシの隠蔽される割合を高くするために外側ループに関するループ展開を行い、中

間語5を出力する。最適化部6は、通常の公知の伝統的な最適化を行い、中間語7を出力する。コード生成部8は、ブリフェッチ命令を含むオブジェクトコードを9を、中間語7をもとに生成する。本発明は4及び8に係わり、オブジェクトコード9の実行効率を向上させるものである。

【0022】図1のループ構造変換部4のうち、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図7に示す。図7に入力するソースプログラム1として、図2のFORTRAN プログラムを例としてあげ説明する。図2のDO20(20), DO10(21)はループを示し、これらは2重ループを構成している。このようなプログラムに対して図7は以下のような処理を行い、中間語5に変換する。

【0023】図7の処理は、ソースプログラム中の多重ループ50を順次処理する。判定51は、多重ループを考慮したブリフェッチコードを出力する多重ループを選択する。ここでは、1つのループの中に、1つのループを含む正規型多重ループに限定する。図2のソースプログラムでは、外側ループDO20(20)の中に、1つのループDO10(21)しかない構造であるので適合する。この例で、外側ループDO20(20)の中に、複数のDOループがある場合は、本発明では、多重ループを考慮したブリフェッチ命令は出力しないので、ループ構造が適合しないと判断する。

【0024】処理52では、元々の内側ループ1回あたりの実行サイクル数を推定する。ループ内には、浮動小数点演算が1個、ロード演算が1個、ストア演算が1個であるので、仮定しているアーキテクチャでは、ループ1回あたり、 $II = \max(\text{ceil}((1+1)/2), \text{ceil}(1/2)) = 1$  サイクルである。

【0025】処理53では、次の外側ループでの処理に必要なデータの主記憶アクセスペナルティを完全に隠蔽する最小のループ長MINLを求める。この場合、主記憶アクセスペナルティは75であるので、MINL=75となる。つまり、このままでは、内側ループ長が75未満であると、次の外側ループの実行に必要なデータをオーバヘッドなしに準備できなく、データ待ちの状態が発生することになる。実際のプログラムにおいては、このループ長よりも小さい場合も多い。

【0026】そこで、仮にコンパイラのコード生成の方針として、ループ長40以上であれば主記憶アクセスペナルティを完全に隠蔽できるコードを生成することにする。判定54において、規定値とは、このようにコンパイラのコード生成方針で決めた値である。図2のソースプログラムの場合、MINL=75、規定値=40であるので、判定55に進む。判定55は、外側ループ展開を行っても、もとのプログラムと同一の処理となるか、データ依存関係からの検査を行う。外側ループ展開が可能か否かは、当該多重ループのループ交換が可能か否か



と同一であり、これは公知技術、例えば田中義一、岩沢京子「ベクトル計算機のためのコンパイル技術」情報処理、Vol.31, No.6, pp.736-743(1990)に記述してある方式を用いればよい。図2のソースプログラムではDO20(20)とDO10(21)のループを交換しても同一の結果を得ることができるので、外側ループ展開が可能である。

〔0027〕処理56では、外側ループの展開数を決めて、外側ループ展開を中間語上で行う。図の例では $\text{ceil}(75/40) = 2$ 倍展開を行い、図8のような結果を得る。ここでは、中間語として、フォートランライクに表現したものをを用いている。DO20(60)とDO10(61)が、外側ループに関して2倍展開した多重ループ、DO11(62)は、外側ループに関して2倍展開した余りループである。以後、余りループに関しては簡単のため、説明は省略する。判定57により、すべての多重ループに対して上記の処理を行う。

〔0028〕その後図1において、通常の間接語に対する最適化部6の処理を行い、コード生成8を行う。コード生成部8において、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図9、図10に示す。

〔0029〕処理90は、最内側ループ内のプリフェッチ対象ロードオペランドの選択を行う。選択の方法は、例えば、公知技術である前記文献ACM 0-89791-535-6/92/0010/0062 pp.62-73に従う。この結果、図8のDO10内の $b(i, j)$ と $b(i, j+1)$ がプリフェッチ対象オペランドとなるとする。

〔0030〕処理91は、すべてのプリフェッチ対象オペランドについてプリフェッチ命令を挿入すべきループ間隔を求め、それらの最大公約数を求める。キャッシュラインサイズが32バイトであるため、 $b(i, j)$ をプリフェッチすると、 $b(i, j)$ を含む連続32バイトのデータがプリフェッチされる。仮に、 $b(i, j)$ がキャッシュラインの先頭にあれば、 $b(i, j)$ 、 $b(i+1, j)$ 、 $b(i+2, j)$ 、 $b(i+3, j)$ のデータプリフェッチされるので、ループ4回に1回の割合でプリフェッチ命令を発行すればよい。同様に、 $b(i, j+1)$ もループ4回に1回でよい。 $b(i, j)$ 及び $b(i, j+1)$ に対するプリフェッチ命令を挿入すべき間隔はそれぞれ4であるため、全体で、プリフェッチ命令を発行するループ間隔は上記の最大公約数であるLC=4となる。

〔0031〕処理92では、最内側ループのループ範囲を2つに分割する。この処理を記述した箱のなかに記すように、ループ後半のループ長は、実行時間を主記憶アクセスペナルティ以上にするループ長か、または、ループ長が短く、主記憶アクセスペナルティ以上の実行時間にならないときはもとのループ長を設定し、ループ前半の処理は残りとする。従って、前半ループは実行さ

れないこともある。

〔0032〕図8のDO10(61)のループ1回あたりの実行推定サイクルは $2 (= \max(\text{ceil}((2+2)/2), \text{ceil}(2/2)))$ 、LC=4、プリフェッチ命令は $b(i, j)$ と $b(i, j+1)$ に対して出るので、プリフェッチ命令による実行サイクルは1である。このため、最内側ループをLC回実行した場合の推定実行サイクルは9サイクルとなる。このため、後半ループのループ長は、 $\min(\text{元々のループ長}, \text{ceil}(75/9))$ となる。その結果、図8のDO10(61)は、図11の前半ループDO10(65)、後半ループDO11(66)に分割できる。

〔0033〕図11では、前半ループDO10(65)、後半ループDO11(66)を、DOループの構造で示してある。このままであると、実際のオブジェクトコードとのレベルギャップがあるので、最内側ループをマシン語レベルに近くした中間語表現を図12に示す。ここで、文70-文78が図11での前半ループDO10(65)に対応し、文80-88が図11での後半ループDO11(66)に対応する。

〔0034〕ここで、文71、文72、文73、文76、文77、文78でDOループ10の処理を意味する。変数cntに処理すべきループ処理長を71で設定し、文76で1つ減じ、文77で繰り返す。文74はもとのループ本体での処理、文70はループ制御変数の初期設定で、文75がループ制御変数の更新を意味する。

〔0035〕同様に、文81、文82、文83、文86、文87、文88でDOループ11の処理を意味する。変数cntに処理すべきループ処理長を81で設定し、文86で1つ減じ、文87で繰り返す。文84はもとのループ本体での処理、文80はループ制御変数の初期設定で、文85がループ制御変数の更新を意味する。

〔0036〕図10に進んで、処理93は、前半ループに対するコード生成を行う。まず、内側ループ後半のためのプリフェッチ命令の挿入を行う。図13の文100、文101、文102がこのコードである。文100により、内側ループで $\text{ceil}(75/9)$ 回先に必要となるデータのアドレスを計算する。文101がプリフェッチ命令で、文102が、LC回おきにプリフェッチコードを出すためのアドレス更新コードである。次にループボディコードを3(=LC-1)回複製して、挿入する。図12でのループボディ文74-77を複製し、文103-文106、文107-文110となる。ループ終了判定コード文77、文106、文110の飛び先のlab1(文73)から、ループ直後のlab2(文78)に変更する。そして、再度、ループボディのコードを複製し、文111-114に挿入する。

〔0037〕処理94は、後半ループに対するコード生

成を行う。前半ループとの違いは、前半ループでは、プリフェッチ対象オペランドの先頭アドレスが、当該外側ループでの内側ループ処理の後半で必要になるデータに対し、1つ先の外側ループの内側ループ処理前半で必要となるデータである点である。最初に、次の外側ループで必要になるデータののためのプリフェッチ命令の挿入を行う。図14の文120,文121,文122がこのコードである。文120により、次の外側ループで必要となるデータのアドレスを計算する。文121がプリフェッチ命令で、文122が、LC回おきにプリフェッチコードを  
10 出すためのアドレス更新コードである。次にループボディコードを3(=LC-1)回複製して、挿入する。図12でのループボディ文84-87を複製し、文123-文126,文127-文130となる。ループ終了判定コード文87,文126,文130の飛び先のlab3(文83)から、ループ直後のlab4(文88)に変更する。そして、再度、ループボディのコードを複製し、文131-134に挿入する。

【0038】処理95は、多重ループの入り口に、初期プリフェッチ命令を挿入する。図13の文140がこれ  
20 である。すなわち、最初の外側ループ繰り返し時における、最初の内側ループの最初に必要となるデータは、ここでプリフェッチをかける。しかし、このプリフェッチは、他の処理とオーバーラップができないので、主記憶ペナルティが隠蔽できない部分である。図6のj=1, i=1, 2, 3, 4, 5のプリフェッチ処理に相当する処理である。

【0039】

【発明の効果】本発明のコード生成方式により、多重ループにおいて、アクセスするデータ領域がキャッシュ容量を超える場合も、内側ループ処理を2つのループに分け、前半ループに必要なデータは、1つ前の外側ループの繰り返し時の後半ループでデータのプリフェッチを開始し、後半ループに必要なデータは、前半ループでデータのプリフェッチを開始するため、主記憶ペナルティを隠蔽することができる。

【0040】例えば従来の手法では、先に述べたように、外側ループ長Mに依存せず、内側ループ長Nが100の場合、全体の時間の内40%がデータ待ちの状態、内側ループ長Nが50の場合は、全体の時間の内56%  
40 がデータ待ちの状態であった。これに対して本発明の実\*

\*施によれば、最内側ループ長が9(=ceil(75/9))以上であれば、外側ループの1回目の処理を除き、2回目以降の処理では、完全に主記憶ペナルティを隠蔽できる。即ち、内側ループ長が100, 50のケースとも、外側ループ長が非常に大きければ、データ待ちの状態は発生しない。外側ループ長が50の場合、内側ループ長が100, 50の場合は、全体の時間の内、1.3%, 2.6%がデータ待ちの状態、外側ループ長が10の場合、内側ループ長が100, 50の場合は、全体の時間の内、6.25%, 11.8%がデータ待ちの状態と改善することができる。

【図面の簡単な説明】

【図1】コンパイラ全体の構成図。

【図2】ソースプログラム例を示す図。

【図3】従来法による、内側ループ展開後のコード例を示す図。

【図4】従来法による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

【図5】従来法による実行の様子を示した説明図。

【図6】本発明による実行の様子を示した説明図。

【図7】本発明による多重ループの外側ループの展開方式の処理フロー図。

【図8】本発明による外側ループ展開後のコード例を示す図。

【図9】本発明による多重ループにおけるプリフェッチコード生成処理のフロー図。

【図10】本発明による多重ループにおけるプリフェッチコード生成処理のフロー図。

【図11】本発明によるループ分割後のコード例を示す  
30 図。

【図12】本発明によるループ分割後の機械語に近いコード例を示す図。

【図13】本発明による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

【図14】本発明による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

【符号の説明】

1…ソースプログラム、2…構文解析部、3…中間語、4…ループ構造変換部、5…中間語、6…最適化部、7…中間語、8…コード生成部、9…オブジェクトコード。

【図2】

図2

```

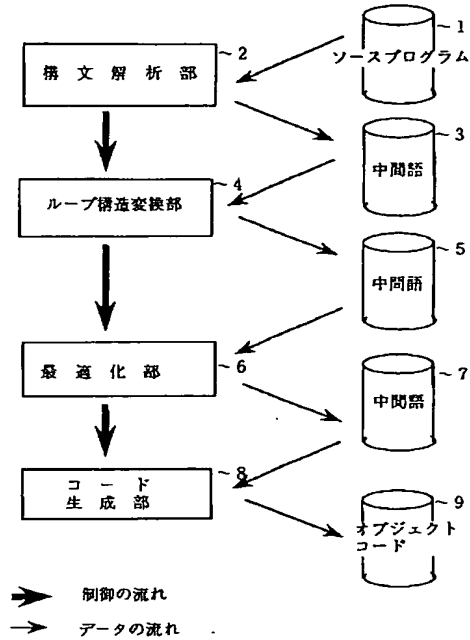
DO 20 j=1, M
DO 10 i=1, N
    a(i, j) = b(i, j) + a
10 continue
20 continue

```

22  
20  
21

【図1】

図1



【図3】

図3

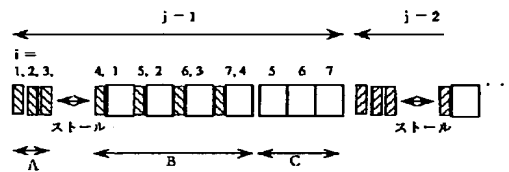
```

DO 20 j=1, M
DO 10 i=1, N-3, 4
  a(i, j)=b(i, j)+s
  a(i+1, j)=b(i+1, j)+s
  a(i+2, j)=b(i+2, j)+s
  a(i+3, j)=b(i+3, j)+s
10 continue
DO 11 i=1+(N/4)*4, N, 4
  a(i, j)=b(i, j)+s
11 continue
20 continue

```

【図5】

図5



【図4】

図4

```

DO 20 j=1, M
DO 1 i=1, min(N, 4*15), 4
  prefetch b(i, j)
1 continue
DO 10 i=1, N-3-4*15, 4
  prefetch b(i+4*15, j)
  a(i, j)=b(i, j)+s
  a(i+1, j)=b(i+1, j)+s
  a(i+2, j)=b(i+2, j)+s
  a(i+3, j)=b(i+3, j)+s
10 continue
DO 11 i=max(1, (N-4*15)/4)*4, N-3, 4
  a(i, j)=b(i, j)+s
  a(i+1, j)=b(i+1, j)+s
  a(i+2, j)=b(i+2, j)+s
  a(i+3, j)=b(i+3, j)+s
11 continue
20 continue

```

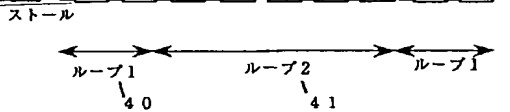
【図6】

図6

```

j=1,1,1,1,1 1,1 1,1 2,1 2,1 2,1 2,1 2,1 2,2 2,2
i=1,2,3,4,5 6,1 7,2 1,3 2,4 3,5 4,6 5,7 6,1 7,2

```



[Shaded Box] プリフェッチ処理 (j=1)    [Shaded Box] プリフェッチ処理 (j=2)  
 [White Box] 演算処理

【図8】

図8

```

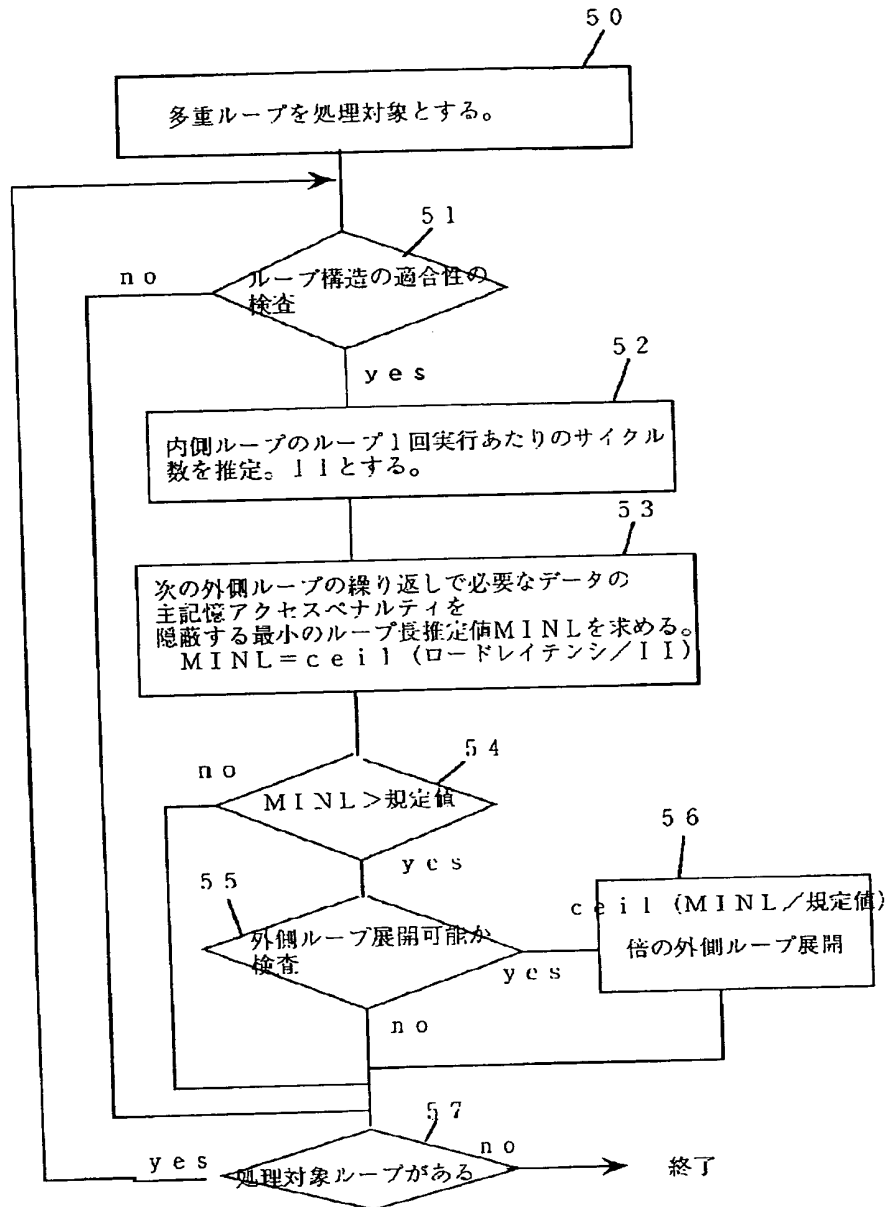
DO 20 j=1, M-1, 2
DO 10 i=1, N
  a(i, j)=b(i, j)+s
  a(i, j+1)=b(i, j+1)+s
10 continue
20 continue

if(mod(M, 2).ne.0) then
DO 11 i=1, N
  a(i, M)=b(i, M)+s
11 continue
endif

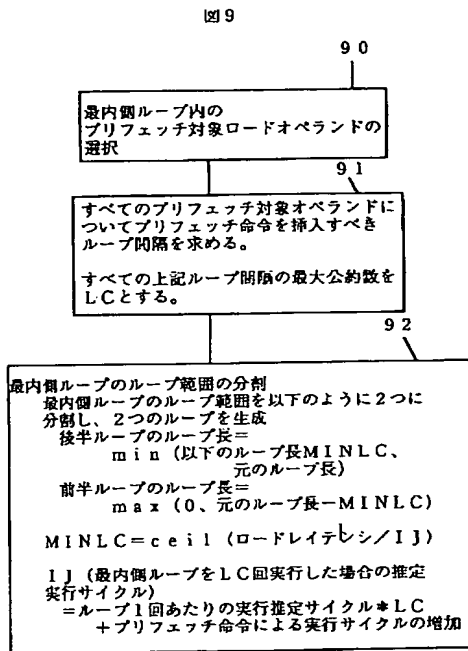
```

【図7】

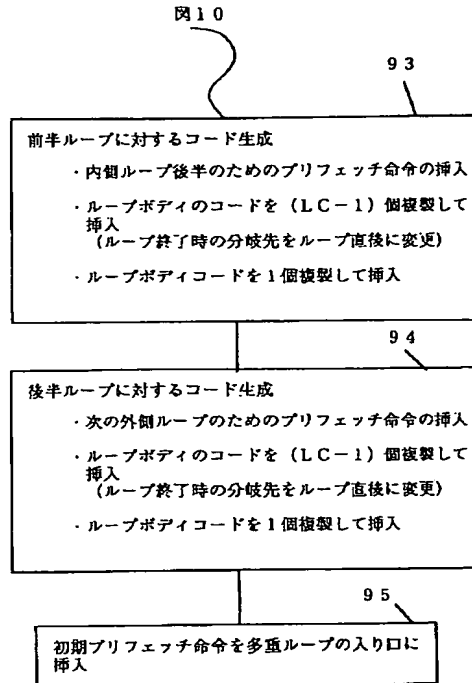
図7



【図9】



【図10】



【図11】

図11-1

```

DO 20 j=1, M-1, 2
DO 10 i=1, N-ceil (75/9)
a (i, j) = b (i, j) + s
a (i, j+1) = b (i, j+1) + s
10 continue
DO 11 i=
max (1,
N-ceil (75/9) + 1), N
a (i, j) = b (i, j) + s
a (i, j+1) = b (i, j+1) + s
11 continue
20 continue
  
```

【図12】

図12

```

DO 20 j=1, M-1, 2
i=1
cnt=N-ceil (75/9) -1
if (cnt=0) goto lab2
lab1:
a (i, j) = b (i, j) + s
a (i, j+1) = b (i, j+1) + s
i=i+1
cnt=cnt-1
if (cnt # 0) goto lab1
lab2:
i=max (1, N-ceil (75/9))
cnt=N-max (1,
N-ceil (75/9) + 1) + 1
if (cnt=0) goto lab4
lab3:
a (i, j) = b (i, j) + s
a (i, j+1) = b (i, j+1) + s
i=i+1
cnt=cnt-1
if (cnt # 0) goto lab3
lab4:
20 continue
  
```

【圖 14】

图 14

```

      5
      14
      i = max (1, N - 1 ceil (75/9)) ~80
      cnt = N - max (1, N - 1 ceil (75/9) + 1) + 1 ~81
      k = 1 ~120
lab3: if (cnt = 0) goto lab4 ~82
      prefetch b (k, j+2) ~83
      prefetch b (k, j+3) ~121
      k = k + 4 ~122
      a (i, j) = b (i, j) + s
      a (i, j+1) = b (i, j+1) + s ~84
      i = i + 1
      cnt = cnt - 1 ~85
      if (cnt = 0) goto lab4 ~87
      a (i, j) = b (i, j) + s
      a (i, j+1) = b (i, j+1) + s ~123
      i = i + 1
      cnt = cnt - 1 ~124
      if (cnt = 0) goto lab4 ~126
      a (i, j) = b (i, j) + s
      a (i, j+1) = b (i, j+1) + s ~127
      i = i + 1
      cnt = cnt - 1 ~128
      if (cnt = 0) goto lab4 ~129
      a (i, j) = b (i, j) + s
      a (i, j+1) = b (i, j+1) + s ~131
      i = i + 1
      cnt = cnt - 1 ~132
      if (cnt = 0) goto lab3 ~133
lab4: ~134
      ~88

```

20 continue